Collab-Board Pre-Search Document

Project: Real-Time Collaborative Whiteboard with AI Agent

Date: February 16, 2026

Author: Michael Habermas

Status: Complete - Ready for MVP Implementation

Executive Summary

Building a production-scale collaborative whiteboard (Miro-like) with an AI agent capable of manipulating board state via natural language. This document establishes the architecture, constraints, and technology choices required to meet the 24-hour MVP gate and final Sunday deadline.

Core Philosophy: A simple success is better than a complicated failure. Single-backend architecture, split state management by frequency (React for slow state, imperative for fast state), and aggressive cost optimization.

Phase 1: Define Your Constraints

1. Scale & Load Profile

- **Launch:** 5-20 users (demo/evaluation audience)
- **6 months:** Unknown, architecture supports 100+ concurrent with horizontal scaling (Socket.io Redis adapter)
- **Traffic:** Spiky (demo day bursts, zero-to-twenty instantly)
- **Real-time:** WebSocket mandatory (Socket.io) for <50ms cursor latency
- **Cold start:** Zero tolerance. Always-on Render service (no serverless functions)

5+ Concurrent Users Without Degradation

- **Definition of "without degradation":** Cursor sync <50ms, object sync <100ms, canvas 60fps (no sustained frame drops), AI response <2s.
- **Verification:** Playwright E2E with 5+ browser contexts on one board (cursor move, object create/move, AI command); optional network throttling; document Render free-tier single-instance limits and upgrade path.
- **Mitigations:** Cursor broadcast throttled to 30fps; object DB writes batched (e.g. 100ms); Socket.io rooms per board; cursors ephemeral in memory (no DB round-trip).

2. Budget & Cost Ceiling

- **Monthly limit:** $0 (free tiers only) for development and launch
- **Infrastructure:** Render free tier (web service), MongoDB Atlas M0 (512MB), Clerk (10k MAU free)
- **AI:** Google Gemini 2.0 Flash (pay-as-you-go, estimated <$1 for development week)
- **Tradeoffs:** Will upgrade to Render Standard ($7/month) only if free tier sleep causes issues during testing

3. Time to Ship

- **MVP Deadline:** 24 hours (hard gate)
- **Final Deadline:** 7 days (Sunday 10:59 PM CT)
- **Priority:** Speed-to-market with interface-driven architecture for swapability
- **Iteration:** Daily commits, vertical slicing (cursor sync → object sync → AI commands)
- **Maintainability:** Clean architecture (SyncEngine abstraction), TDD for critical sync logic

4. Compliance & Regulatory Needs

- **None.** Class project environment. No HIPAA, GDPR, SOC2, or data residency requirements.

5. Team & Skill Constraints

- **Team:** Solo (human + AI coding agents)
- **Known Stack:** React, TypeScript, Vite, Bun, Tailwind v4, shadcn/ui
- **New Technologies:** Socket.io, Clerk, Google Gemini API, Konva.js, MongoDB (familiar but not expert)
- **Learning Appetite:** High for AI tools, low for infrastructure complexity (managed services preferred)
- **AI Tools:** Cursor (primary IDE) + Claude Code (secondary CLI agent)

Phase 2: Architecture Discovery

6. Hosting & Deployment

- **Platform:** Render (monolith deployment)
- **Pattern:** Single service serving:

1. Static React build (frontend)
2. WebSocket upgrade endpoint (real-time sync)
3. REST API (AI commands)

- **Runtime:** Bun (instead of Node.js) for 30% faster cold starts and native TypeScript
- **CI/CD:** Git push to main → Render auto-deploy (Bun build ~30s)
- **Scaling:** Vertical scaling on Render (Pro tier), horizontal via Socket.io Redis adapter (future)

7. Authentication & Authorization

- **Service:** Clerk (React SDK + Node SDK)
- **Methods:** Magic links + Google OAuth (fastest UX, no password management)
- **WebSocket Security:** JWT verification on Socket.io handshake (`socket.auth.token`)
- **RBAC:** Board-level permissions (owner/collaborator) stored in MongoDB, enforced at API level
- **Multi-tenancy:** Isolated by `boardId` (Socket.io rooms), application-level access control

8. Database & Data Layer

- **Type:** Document store (MongoDB Atlas)

- **Why MongoDB:** Schemaless board objects (sticky notes, shapes, frames have divergent properties), zero migrations during rapid iteration week
- **Schema:**

```
boards: { _id, title, ownerId, collaborators[], createdAt }
objects: { _id, boardId, type, x, y, width, height, content, color, createdBy, updatedAt }
```

- **Read/Write Pattern:**
- **Heavy write:** Cursor updates (handled in Socket.io memory, never hits DB)
- **Moderate write:** Object CRUD (throttled persistence to MongoDB)
- **Read:** Board load on connection (single query by `boardId`)

9. Backend/API Architecture

- **Pattern:** Monolith (Express.js with Bun)
- **API Styles:**
- **REST:** `/api/ai/execute` for AI commands (synchronous, <2s response)
- **WebSocket:** Socket.io events for cursors, objects, presence (real-time, <50ms)
- **Background Jobs:** None (Gemini Flash fast enough for synchronous responses)

10. Frontend Framework & Rendering

- **Framework:** React 18 (Strict Mode)
- **Build Tool:** Vite + Bun
- **Rendering:** SPA (Single Page Application) - no SSR needed for whiteboard app
- **Canvas:** Konva.js with react-konva for React integration
- **Styling:** Tailwind CSS v4 + shadcn/ui (latest components)
- **State Management:**
- **Zustand:** Auth, board metadata, object list (low-frequency updates)
- **Direct Konva refs:** Cursor positions (high-frequency, 30-60fps, bypasses React reconciliation)

11. Third-Party Integrations

- **Auth:** Clerk (10k MAU free, vendor lock-in acceptable for MVP, swappable to Lucia later)
- **AI:** Google Gemini 2.0 Flash (OpenAI-compatible endpoint, 1M context window, 90% cheaper than GPT-4)
- **Database:** MongoDB Atlas (M0 free tier, 512MB storage)
- **Hosting:** Render (free web service tier)
- **API Documentation:** Swagger/OpenAPI (swagger-ui-express) auto-generated from Express routes
- **Risks & Mitigations:**
- **Gemini rate limits:** 1,500 req/min free tier (sufficient for MVP, monitor at scale)
- **Render sleep:** Free tier sleeps after 15min inactivity (mitigate with uptime ping or upgrade to $7/month)
- **Vendor lock-in:** Clerk and Gemini use standard protocols (JWT, OpenAI API spec), easy to swap

---

Phase 3: Post-Stack Refinement

12. Security Vulnerabilities

- **WebSocket Authorization:** Verify Clerk JWT on Socket.io `connection` event, reject unauthorized before allowing room join
- **MongoDB Injection:** Use Zod validation on all inputs, never concatenate user input into queries
- **CORS:** Restrict to specific domains in production (Render env vars)
- **AI Prompt Injection:** Sanitize user input before sending to Gemini (no system prompt leakage via user content)
- **Dependency Risks:** Pin versions in `bun.lockb`, audit with `bun audit` before deployment

13. File Structure & Project Organization

Monorepo with Bun Workspaces:

```
collabboard/
■■■ apps/
■   ■■■ client/              # React + Vite (Bun runtime)
■   ■   ■■■ src/
■   ■   ■   ■■■ components/  # UI components (shadcn) + Canvas components (Konva)
■   ■   ■   ■■■ hooks/       # useSocket, useBoard, useAuth
■   ■   ■   ■■■ lib/         # Clerk client, Socket client, utils
■   ■   ■   ■■■ store/       # Zustand stores (boardStore, authStore)
■   ■   ■   ■■■ types/       # Shared TypeScript interfaces
■   ■   ■■■ package.json
■   ■■■ server/              # Bun + Express + Socket.io
■       ■■■ src/
■       ■   ■■■ handlers/    # Socket event handlers (cursor, object, presence)
■       ■   ■■■ routes/      # REST routes (AI execution)
■       ■   ■■■ lib/         # DB connection (MongoDB), Clerk verification
■       ■   ■■■ types/       # Shared TypeScript interfaces
■       ■■■ package.json
■■■ packages/
■   ■■■ shared-types/        # Shared between client/server (optional)
■■■ package.json             # Root workspace config
```

14. Naming Conventions & Code Style

- **Language:** TypeScript (strict mode, `noImplicitAny`)
- **Naming:** PascalCase components/hooks, camelCase variables/functions, UPPER_SNAKE_CASE constants
- **Linting:** ESLint (flat config) + Prettier
- **Imports:** Absolute imports via `~/` alias (Vite tsconfig paths)
- **File Naming:** Kebab-case for files (e.g., `cursor-handler.ts`), PascalCase for components

15. Testing Strategy

- **Unit:** Vitest for utility functions (coordinate math, AI command parsing)
- **Integration:** Test Socket.io handlers with `socket.io-client` + memory MongoDB (mongodb-memory-server)
- **E2E:** Playwright for multiplayer scenarios (opens two browser contexts, tests real-time sync, network throttling, disconnect recovery)
- **Coverage Target:** 60% for MVP (focus on sync logic and AI command handlers)
- **Mocking:** MSW for Clerk auth in tests, mock Gemini responses for AI handler tests

16. Recommended Tooling & DX

- **IDE:** Cursor (primary) + Claude Code (CLI secondary)
- **VS Code Extensions:**
- ESLint, Prettier, Tailwind CSS IntelliSense
- Bun for VS Code (debugging support)
- **API docs/testing:** Swagger/OpenAPI (swagger-ui-express) at `/api-docs`
- **CLI Tools:**
- `bun` (package manager + runtime + test runner)
- `mongosh` (MongoDB shell for debugging)
- **Debugging:**
- React DevTools (Zustand integration)
- Socket.io Admin UI (monitor rooms/connections)
- Swagger UI at `/api-docs` (auto-generated API documentation)
- Render Dashboard logs (streaming)

Appendix Checklist — Explicit Answers

Phase 1–3 checklist in appendix format (each sub-question answered for grading).

1. Scale & Load Profile — Users at launch? 5–20 (demo/evaluation). In 6 months? Unknown; architecture supports 100+ with horizontal scaling. Traffic pattern? Spiky (demo day bursts). Real-time requirements? WebSocket (Socket.io) mandatory for <50ms cursor latency. Cold start tolerance? Zero; always-on Render service.

2. Budget & Cost Ceiling — Monthly spend limit? $0 (free tiers only). Pay-per-use acceptable? Yes for AI (Gemini pay-as-you-go). Where trade money for time? Upgrade Render to $7/month if free-tier sleep causes issues.

3. Time to Ship — MVP timeline? 24 hours (hard gate), final 7 days. Speed vs. maintainability? Speed-to-market with interface-driven swapability. Iteration cadence? Daily commits, vertical slicing.

4. Compliance & Regulatory Needs — HIPAA/GDPR/SOC 2/data residency? None; class project.

5. Team & Skill Constraints — Solo (human + AI agents). Known stack: React, TypeScript, Vite, Bun, Tailwind v4, shadcn. New: Socket.io, Clerk, Gemini, Konva, MongoDB. Learning appetite: high for AI tools, low for infra complexity.

6. Hosting & Deployment — Serverless vs. containers vs. edge vs. VPS? Single Render web service (VPS-style, always-on). CI/CD? Git push to main → Render auto-deploy. Scaling? Vertical on Render; horizontal via Socket.io Redis adapter later.

7. Authentication & Authorization — Auth approach? Clerk (magic links + Google OAuth). RBAC? Board-level (owner/collaborator) in MongoDB. Multi-tenancy? By `boardId` (Socket.io rooms).

8. Database & Data Layer — Type? Document store (MongoDB Atlas). Real-time sync? Via Socket.io (cursors in memory); persistence to MongoDB for objects. Full-text/vector/caching? None for MVP. Read/write ratio? Heavy write on cursors (ephemeral), moderate on objects, light read on board load.

9. Backend/API Architecture — Monolith or microservices? Monolith (Express + Bun). REST vs. GraphQL etc.? REST for AI (`/api/ai/execute`), WebSocket for real-time. Background jobs? None.

10. Frontend Framework & Rendering — SEO? Not required (whiteboard SPA). Offline/PWA? No. SPA vs. SSR? SPA; Konva canvas, Tailwind v4, shadcn.

11. Third-Party Integrations — Services? Clerk, Gemini, MongoDB Atlas, Render. Pricing cliffs/rate limits? Gemini 1,500 req/min; Render free tier sleeps 15min. Vendor lock-in? Mitigated via standard protocols (JWT, OpenAI-compatible API).

12. Security Vulnerabilities — Pitfalls? WebSocket auth (verify JWT on connection). Misconfigurations? CORS restrict in production. Dependency risks? Pin versions, `bun audit` before deploy; Zod for input validation; sanitize AI input.

13. File Structure & Project Organization — Structure? Monorepo, Bun workspaces (`apps/client`, `apps/server`). Monorepo vs. polyrepo? Monorepo. Feature organization? components, hooks, lib, store, types (client); handlers, routes, lib, types (server).

14. Naming Conventions & Code Style — Naming? PascalCase components/hooks, camelCase vars/functions, UPPER_SNAKE_CASE constants. Linter/formatter? ESLint (flat config) + Prettier. Imports? Absolute via `~/` alias.

15. Testing Strategy — Unit/integration/e2e? Vitest (unit), socket.io-client + memory MongoDB (integration), Playwright (e2e multiplayer). Coverage target? 60% for MVP. Mocking? MSW for Clerk, mock Gemini for AI tests.

16. Recommended Tooling & DX — IDE: Cursor + Claude Code. Extensions: ESLint, Prettier, Tailwind IntelliSense, Bun for VS Code. CLI: `bun`, `mongosh`. Debugging: React DevTools, Socket.io Admin UI, Swagger at `/api-docs`, Render logs.

Key Architecture Decisions (Defensible)

Decision: Socket.io + MongoDB over Firebase/Supabase

Why: Firebase Firestore has 100-300ms latency (violates <50ms cursor spec). Socket.io on Render provides 10-30ms cursor updates with persistent connections. MongoDB provides schemaless flexibility for rapid iteration.

Alternatives considered: Firebase/Supabase (rejected: 100–300ms latency); Firestore-only (rejected: cannot hit <50ms cursors). Database: PostgreSQL considered (rejected: schema migrations for divergent object types); tradeoff: schemaless MongoDB for iteration speed.

Decision: Zustand for Slow State + Direct Konva for Fast State

Why: Context API causes app-wide re-renders at 30fps (performance death). Zustand isolates subscriptions to specific components. Cursors bypass React entirely (direct Konva ref updates) to achieve 60fps and <50ms sync without reconciliation overhead.

Alternatives considered: Context-only (rejected: 60fps re-renders kill performance); Zustand-for-everything including cursors (rejected: frame drops at 5 users). Tradeoff: mixed pattern (right tool per update frequency).

Decision: Google Gemini 2.0 Flash over GPT-4/Groq

Why: 90% cost reduction ($0.075 vs $0.70+ per 1M tokens), 1M token context window (can send full board state for complex layout commands), OpenAI-compatible API (easy migration), sub-second latency for simple commands.

Alternatives considered: Groq (cheaper initially, rejected for Gemini's cost/context); GPT-4 (rejected: cost). Tradeoff: Google data processing acceptable for class project; rate limits monitored at scale.

Decision: Monolith over Microservices

Why: Single deploy target reduces complexity for 24h MVP. Eliminates CORS issues, network latency between services, and deployment coordination. WebSocket server + API + static hosting in one process.

Alternatives considered: Split (e.g. Vercel frontend + Render backend) rejected: CORS and two deploy surfaces. Tradeoff: single process for 24h MVP; can split later if needed.

Decision: Clerk over Firebase Auth/Custom Auth

Why: Pre-built React components save 4-6 hours of UI development. JWT verification middleware works seamlessly with Socket.io. 10k MAU free tier sufficient for launch.

Alternatives considered: Lucia/custom auth (rejected: 4–6h to build login UI and session management). Tradeoff: vendor lock-in acceptable for MVP; swappable later (JWT standard).

Decision: Optimistic UI with Rollback

Why: For object creation/movement, render immediately on client (perceived <50ms latency), emit to server, roll back only on error acknowledgment. Better UX than waiting for MongoDB round-trip.

Alternatives considered: Server-confirmation-only (rejected: latency). Tradeoff: possible brief inconsistency on error; rollback handles it.

Decision: Bun over Node.js + npm

Why: 30% faster cold starts, native TypeScript support (no ts-node), single lockfile, built-in bundler. Eliminates `node_modules` bloat during rapid iteration.

Alternatives considered: Node.js + npm (rejected: slower cold starts, extra tooling). Tradeoff: smaller ecosystem than Node for some packages; sufficient for this stack.

AI Cost Analysis (Required Submission Component)

Development & Testing Costs (Estimated)

- **Model:** Google Gemini 2.0 Flash
- **Pricing:** $0.075/1M input tokens, $0.30/1M output tokens
- **Development Estimate:**
- ~500 AI calls during development/testing
- ~800 tokens average per call (mix of simple and complex commands)
- **Total Dev Cost:** ~$0.50

Production Cost Projections

Assumptions:

- 20 AI commands per user session (mix of simple creation + complex templates)
- 800 tokens per command average (input + output)
- 10 sessions per user per month
- Gemini pricing: $0.075/1M input, $0.30/1M output

Users	Monthly AI Cost	Monthly Infra Cost	Total Monthly
100	$1.50	$0 (Render free + MongoDB M0 + Clerk free)	**$1.50**
1,000	$15	$0 (Render free + MongoDB M0 + Clerk free)	**$15**
10,000	$150	$85 (Render Standard $7 + MongoDB M10 $60 + Clerk $25*)	**$235**
100,000	$1,500	$400 (Render Pro $25 + MongoDB M30 $300 + Clerk $75)	**$1,900**

*Clerk free to 10k MAU, then Pro plan includes first 10k for $25/month

---

Requirements Verification Matrix

MVP Requirements (24h Hard Gate)

Requirement	Implementation	Status
Infinite board + pan/zoom	Konva Stage with draggable viewport + scale transform	■
Sticky notes (create, edit text)	`createStickyNote` tool + Konva Text editing	■
Shapes (rect/circle/line)	`createShape` tool + Konva Rect/Circle/Line	■
Create/move/edit objects	Socket.io `object:create`, `object:move` events + optimistic UI	■
Real-time sync (2+ users)	Socket.io rooms (`board:${id}`) broadcast to all clients	■
Multiplayer cursors + names	Direct Konva cursor layer, Socket.io `cursor:move` at 30fps	■
Presence awareness	Socket.io `connection`/`disconnect` events update user list	■
User authentication	Clerk React SDK + JWT verification on Socket.io handshake	■
Deployed publicly	Render monolith auto-deploy on git push	■

Performance Targets

Metric	Target	Implementation	Status
Frame rate	60 FPS	Konva canvas bypasses React render cycle, requestAnimationFrame	■
Object sync	<100ms	Socket.io broadcast (~10-30ms) + optimistic local render	■
Cursor sync	<50ms	Direct Socket.io emit (no DB round-trip), 30fps throttle	■
Object capacity	500+	Konva handles 10k+ nodes; MongoDB indexes on `boardId`	■
Concurrent users	5+	Socket.io connection limit ~1k per Node instance	■

AI Board Agent Requirements

Category	Commands	Tools	Status
Creation	Add sticky, create shape, add frame	`createStickyNote`, `createShape`, `createFrame`	■
Manipulation	Move objects, resize, change color	`moveObject`, `resizeObject`, `changeColor`	■
Layout	Grid arrangement, spacing	`arrangeInGrid` (composite using moveObject)	■
Complex	SWOT template, user journey	Multi-step planning via Gemini function calling chain	■
Schema	9 required functions	`createConnector`, `updateText`, `getBoardState`, etc.	■
Shared AI State	All users see AI results	AI execution broadcasts via Socket.io to room	■
Performance	<2s response, 6+ command types	Gemini Flash latency + comprehensive tool coverage	■

Testing Scenarios

Scenario	Implementation	Status
2 users editing simultaneously	Socket.io rooms with unique user IDs	■
One user refreshing mid-edit	MongoDB `find({boardId})` on client `connect` event	■
Rapid creation/movement	Optimistic UI + throttled DB writes (100ms batch)	■

Scenario	Implementation	Status
Network throttling	Socket.io automatic reconnect with exponential backoff	■
5+ concurrent users	Load testing with Playwright + Render limits documented	■

AI-First Development Requirements

Requirement	Implementation	Status
Tools	Cursor (IDE) + Claude Code (CLI)	■
MCP Usage	Evaluate Cursor MCPs (Git, MongoDB) during build; planned: Git (version control from IDE), MongoDB (data exploration). Document actual usage in AI Development Log.	■
AI Development Log	Document prompts, % AI-generated code, token costs	■
Cost Analysis	Table above + actual spend tracking during dev	■

Pre-Search Checklist Status

- ■ Phase 1: Constraints defined (Scale, Budget, Time, Compliance, Team)
- ■ Phase 2: Architecture discovered (Hosting, Auth, Database, API, Frontend, Integrations)
- ■ Phase 3: Stack refined (Security, File Structure, Naming, Testing, Tooling)
- ■ Key decisions documented with tradeoffs
- ■ AI Cost Analysis completed with projections
- ■ All MVP and project requirements verified

AI conversation (reference document): pre-search-ai-conversation.md. External link: